Supercomputing Visualization for Earth Science Datasets

P. Peggy Li
Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109
Peggy.Li@jpl.nasa.gov

Abstract **– Two Supercomputer-based parallel visualization systems, RIVA and ParVox, are presented in this paper. RIVA (Remote Interactive Visualization and Analysis) System is a parallel terrain rendering system and ParVox (PARallel VOXel renderer) is a parallel volume rendering system for multiple-variable, time-varying, 3D volume datasets in structured grid and unstructured grid. Both systems are designed for interactive visualization of very large scientific datasets on remote supercomputers. The system architectures and the parallel algorithms are described in this paper. In addition, we will use North Atlantic Ocean Model as a sample application to demonstrate how RIVA and ParVox can help scientists to discover and present their scientific results.**

I. INTRODUCTION

There are two main driving forces behind the development of software-based parallel rendering systems: 1. the use of supercomputers for large scale scientific modeling and simulations enables the modelers to generate larger data volumes in greater speed, thus, it becomes impractical to transfer, store, and visualize large volumes of datasets on user's local workstation. Second, high-resolution terrain rendering and volume rendering of large datasets demand both extensive computational and storage resources which a workstation can hardly provide.

Parallel rendering algorithms are mainly based on three different sequential approaches: ray casting, shearing, and forward projection. Ray-casting can be easily parallelized using image space decomposition, but it requires input data set available at every rendering node, thus not suitable for distributed memory parallel architectures. Forward projection is used by most graphics hardware for polygon rendering. It is ideal for object space decomposition and is the most efficient algorithm for rendering very large datasets.

We developed two new parallel rendering algorithms based on the forward projection approach, the whole earth renderer for terrain datasets and the parallel splatting algorithm for 3D volume datasets. We also designed and implemented two distributed visualization systems around these rendering algorithms. In section II, we present the first system, RIVA (Remote Interactive Visualization and Analysis) system, for large terrain data sets [1]. In section III, we present the second system, ParVox (PARallel VOXel Rendering) for time-varying, multi-variable volume data sets [2]. In Section IV, we present an ocean model applica-tion and its visual results using both RIVA and ParVox.

II. RIVA SYSTEM OVERVIEW

The kernel of the RIVA system is a parallel terrain renderer running on parallel supercomputers. The renderer produced 3D perspective views of terrain using earth or planetary images or simulation datasets with coregistered digital elevation data. Because the underlying geometric model is that of a sphere, the renderer can accommodate global datasets; accordingly we refer to this renderer as the *whole earth renderer*.

A. System Architecture

Around the core renderer, RIVA is equipped with a suite of Graphic User Interface (GUI) programs for data navigation, display and animation editing. As depicted in the system architecture diagram in Fig. 1, the main RIVA data navigator program, *Flexible Flyer*, resides on a SGI workstation. A low resolution copy of the dataset is loaded into the Flexible Flyer and user can navigate the dataset and select the desired views. As the user navigates, his/her viewpoint is transmitted to the whole earth renderer residing on a remote supercomputer via a network interface program, *NetHost*. The renderer renders the image using a full resolution copy of the dataset and sends the resulting image back to a display window, *receive_display*, on the user's workstation.

RIVA is designed for both interactive exploration of large datasets and batch generation of animations. The Flexible Flyer has a key frame editor built in where key frames can be inserted, appended, modified, and previewed. A separate 2-D map display window, *xshow*, displays the key frames or the flight path on a 2D map of the data. It helps a user to identify his location and direction in a global orientation and also help a user to select key frames in a more even pace. Once the key frames are selected, the flight path is calculated using a cubic spline algorithm. The renderer then renders the flight path in the batch mode and save the animation frames into disk.

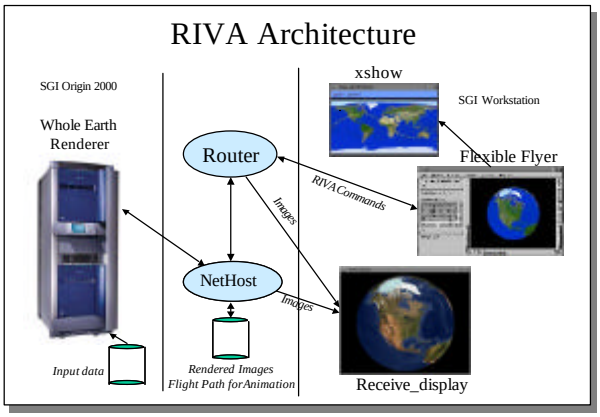

Fig. 1 The RIVA System Architecture

The middle section of Fig. 1 is the network interface programs for RIVA. *NetHost* is the interface (or, host) program between the GUI and the renderer. RIVA uses a text-based command language to communicate with the renderer. NetHost is the one who processes and dispatches commands, receives and distributes results. The commands may come from a network interface, such as a socket stream fed by the GUI program, or from a disk file. The *Router* is a routing daemon that facilitates dynamic, reliable, multi-casting message passing services. It detached the physical connection between the renderer and the GUI program, thus allowing either part of the program to run as a stand alone entity, a renderer to feed multiple displays on different workstations, or even one GUI program to control two parallel renderers.

B. Parallel Algorithm

Rather than using geometric objects (such as triangle strips) to represent the digital terrain as all the hardware-based terrain renderers do, the whole earth renderer represents and renders the terrain pixel by pixel. We use a parallel forward projection rendering algorithm with both object space decomposition and image space decomposition. The detail of the algorithm can be found in [1], here is a brief summary of the algorithm. The input data, both image and elevation in either cylindrical projection or sinusoidal projection, are equally divided into small tiles and distributed to each rendering processor. The processor applies the transformation matrix to its local data and transforms them into image space coordinate. Each rendering processor produces patches of images scattered in the final image. The image patches are then merged and composited into the final image using a binary-swap method. The image tiles can be rendered in any order and there is no communication required in the transformation stage. The rendering processors synchronize globally before the final compositing begins. The rendering speed is determined by the slowest processor in the transformation stage.

Several optimization techniques have been implemented in RIVA to improve the rendering speed and image quality:

Data pyramiding. Also known as mip-mapping. The terrain is rendered at a different resolution based on its distance to the viewpoint. This technique is used to reduce computation as well as eliminate aliasing problem in the far field. We generate data pyramid on the fly to save memory space at the cost of some computation overhead.

Culling. Several culling techniques are used to eliminate the input tiles that fall outside the field of view of a given viewpoint. *Horizon Test* calculates the distance from the viewpoint to the center of each tile and eliminates the tiles that fall behind the horizon. This test can eliminate almost half of the tiles in a global dataset. *Tile Test* eliminates the tiles whose projected areas fall outside the viewport.

C. Functionality

There are several unique features that distinct RIVA from other terrain renderers:

Multiple Data Representations – Internally, RIVA represents the data using a spherical model regardless it is a global dataset or a regular grid dataset. Externally, RIVA can process data in either 2D Cartesian, 3D Cartesian, or 3D Polar coordinates. The dataset can be stored in either Sinusoidal projection to save space or in Cylindrical projection for efficient processing. RIVA is flexible and trying to accommodate different application need and different data representations.

Multiple Surface Rendering – RIVA can render multiple terrain surfaces with different resolution, different data format, and different coverage. The multiple surfaces can be combined using various blending methods. Fig. 2 is an example of alpha-blending of two surfaces. The top surface is a grayscale image of Coronado Island at 2.25 meter resolution (Fig. 2c) and the bottom surface is a color LandSat image at 30 meter resolution (Fig. 2a). By setting the opacity of the top surface to 0.58, it gives you a colored image at 2.25 meter resolution (Fig. 2b). Fig. 3 is an example of zbuffer compositing. The top surface is a North Atlantic Ocean surface with color representing ocean surface temperature. The bottom surface is a topographic map of the ocean bottom. The top surface is raised up so that the two surfaces can be separated.

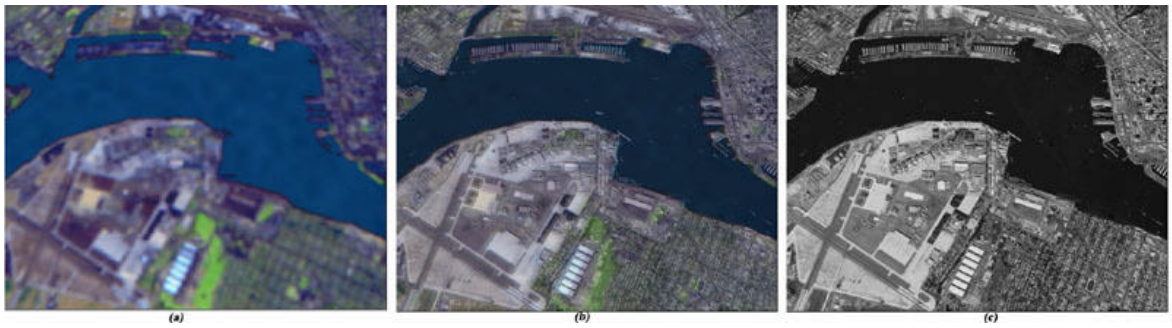

Fig. 2 Multiple Surfaces composited by blending, (a) 30 meter LandSat, (b) two surfaces blended with the 2.25 meter dataset at opacity=0.58, (c) 2.25 meter grayscale dataset

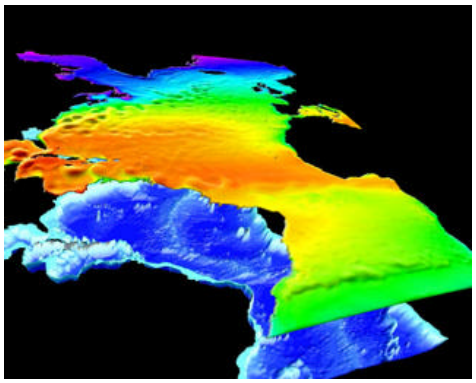

Fig. 3 Multiple Ocean Surfaces composited by zbuffer

Large Dataset Rendering – RIVA allows out-of-core rendering for datasets that exceed the capacity of the physical memory. A lower resolution sample of the original dataset has to be prepared in advance. RIVA loads the lower resolution dataset and renders it until the data pyramiding algorithm identifies that a higher resolution image tile is needed. The full resolution image tile will then be loaded into memory. A memory cache is used to keep the most recent tiles used to reduce disk I/O. RIVA also renders time-varying datasets generated by simulations. Similar to out-of-core rendering, only the data for the first time step resides in the memory. The rest data will be loaded into memory when the animation starts.

High Resolution Animation– RIVA is not only scalable to large input datasets but also scalable to large image outputs. RIVA images are not limited to the framebuffer size or the screen resolution as other terrain renderers do. RIVA can render a large image in multiple passes by partitioning the images into multiple viewports. Theoretically, there is no limit to the image size in RIVA. We have done three screen panoramic movie and animations in HD format.

D. Implementation

RIVA has gone through several evolutions. An earlier version of the algorithm was first developed on Intel Paragon. The RIVA architecture was designed and first implemented on the Cray T3D using its shemem libarary. The Flexible Flyer was implemented using OpenGL and SGI's OpenInventor API. RIVA has been used as a production tool for several animation products in JPL until Cray T3D was decommissioned in 1998. In 2001, RIVA was ported to SGI Origin 2000 using a combination of shared memory and message passing model. New functionalities and new tools were added into RIVA to improve its key frame editing capability and to read input image data in standard image formats. The first RIVA release was made public in 2001.

III. PARVOX SYSTEM OVERVIEW

A. System Architecture

ParVox is a parallel volume rendering system for either distributed visualization, or as a rendering API to be linked with application programs. As a distributed visualization system shown in Fig. 4, ParVox provides an X window based GUI program for display and viewing control, two input modules that read structured and unstructured 4D datasets in NetCDF format, respectively, two core renderers, one for structured grid dataset and one for unstructured grid dataset, and an output module that supports multiple output formats, including wavelet image compression format for both loseless and lossy compressions. The input, the renderer, and the output modules form a functional pipeline using MPI for inter-module communication. As a rendering API, ParVox supplies users with a parallel input library to read 4D structured grid and unstructured grid datasets in NetCDF format, an OpenGL style parallel rendering API for graphic controls and a parallel wavelet image compression library.

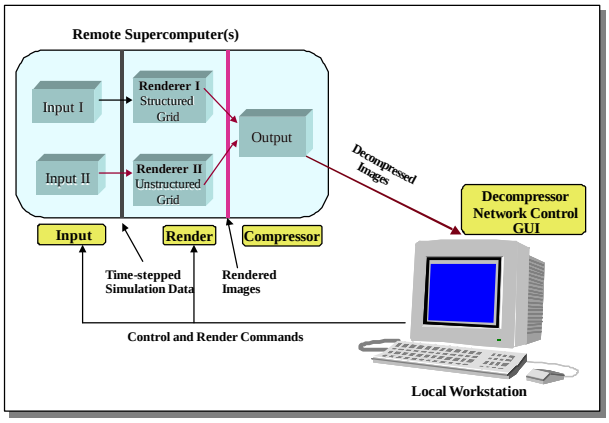

Fig. 4 The ParVox System Architecture

ParVox can visualize 3-D volume data as a translucent volume with adjustable opacity for each different physical value, or as multiple isosurfaces at different thresholds and different opacities. It can also slice through the 3D volume and view only a set of slices in either of the three major orthogonal axes. Moreover, it is capable of animating time-sequence 3D datatsets at any selected viewpoint.

B. Parallel Algorithms

ParVox uses a parallel splatting algorithm for structured grid datasets [2]. Similar to the whole earth renderer, the splatting algorithm is a forward projection algorithm that is ideal for parallel execution with object space decomposition. The input volume is partitioned into small interleaving blocks distributed into each local processor's memory. Each processor first renders its volume blocks locally by splatting and compositing each voxel to the local accumulation buffer. The sub-image is then composited with other sub-images from other processors. The global image compositing is done in parallel by partitioning the image space into small interleaving regions and assigning multiple regions to each of the processors. Communication is required to redistribute the sub-images from the splatting processor to the compositing processor. The splatting can be done in any order, but the image compositing has to be in either the front-to-back or the back-to-front order. Therefore, final compositing cannot start until the splatting process is finished and all the sub-images have been sent to the destination compositing processor. However, the sub-image will be redistributed asynchronously right after it's been generated. By overlapping the splatting and the image redistribution process, a major portion of communication overhead can be hidden.

The unstructured grid volume renderer in ParVox is an adaptation of Ma's cell-projection parallel renderer [3]. The volume is represented as a list of vertices and a list of tetrahedron cells. The cells are evenly distributed into the local memory of the rendering processors in order to achieve better load balancing. A synchronized preprocessing is then performed to organize the local cells into a space partitioning tree. The local cells are partitioned into spatial regions such that each region represents equal amount of rendering load. The cells will be rendered in the order of its region location in all the processors. Each cell is scan converted into ray segments in the image coordinates. The ray segments will then be routed to the merging processor for final merging. The scan conversion and ray-segment redistribution are interleaved in a similar fashion as in the splatting algorithm. The ray-segment can be merged as long as the adjacent rays are both present. Therefore, the spatial partitioning tree allows the cells in the neighborhood be rendered about the same time, thus reducing the memory usage and improving the overall efficiency. When the scan conversion and ray-segment merging are finished, each processor sends its completed sub-image to the output module where the final image is assembled, compressed and sent out to display.

C. Functionalities

ParVox was designed for distributed visualization assuming the end user has a low bandwidth network connection and a limited function workstation at his desktop. It also assumes that the user has access to a remote supercomputer and has his dataset residing on the supercomputer's disk. ParVox has unique features to facilitate interactivity in such an environment:

Functional Pipelining-- ParVox contains three modules, the input module, the rendering module, and the output module. Each module can be run in parallel and three modules are connected as a functional pipeline using MPI to communicate with each other. The functional pipeline allows overlapping data input, rendering, and image compression. We can use different number of processors for each module to balance the load of each module.

Parallel Wavelet Compression – A parallel implementation of ERIC (Efficient resersible Image Compression) [4] algorithm is used in ParVox for image compression. It supports both loseless and lossy compression and it allows for progressively accurate approximation based on the network bandwidth. Empirical results show that a low-frequency image can preserve its image quality with insignificant degradation at a compression ration as high as 50.

X Window Based User Interface – The ParVox GUI was designed using X/Motif and runs on any Unix workstation. It provides a user-friendly, interactive environment for viewing and rendering control. Multiple control panels are provided to control the direct rendering, iso-surface classification, viewing transformation, lighting

and material setting. ParVox also integrates the color and opacity editing program, icol [5] developed by AHPCRC, University of Minisota. Animation control and ins tant playback is also built in for animation . The rendering parameters can be saved into a NetCDF file and be restored later or used for batch-mode processing.

D. Implementation

The core renderer of ParVox was first implemented on the Cray T3D and Cray T3E using shmem onesided communcation API. It was later ported to MPI and runs on SGI Origin 2000, HP Exemplar System and Beowulf Clusters. The ParVox GUI runs on any system capable of running X window. Two ParVox system has been released for public use. The document is available at http://alphabits.jpl.nasa.gov/ParVox .

IV. APPLICATIONS

RIVA has been used as a production tool to produce animations using LandSat and Planetary datasets for publ ic outreach and scientific education. ParVox has been used to visualize various time -varying, multiple -parameter, 3D datasets. In this paper, we will use ocean modeling application to demonstrate how these two visualization tools help scientists to disco ver scientific results from their massive modeling data.

Ocean modeling has played an important role in obtaining comprehensive understanding of world ocean circulation, monitoring current climatic conditions, and predicting future climate changes. JPL has conducted a 40-year simulation of a 1/6 degree ocean model [6] for the North Atlantic Ocean using Parallel Ocean Program (POP) [7]. The total data generated by this model is 2.8 Terabytes assuming a snapshot is saved for every three simulation days. The model generates both 2D data, such as surface temperature and surface height, and 3D data, such as tem perature, salinity, and velocity.

We used RIVA to visualize the 2D ocean surface data and ParVox to visualize the 3D volume data. In Fig. 6b, the ocean surface temperature data is represented using a

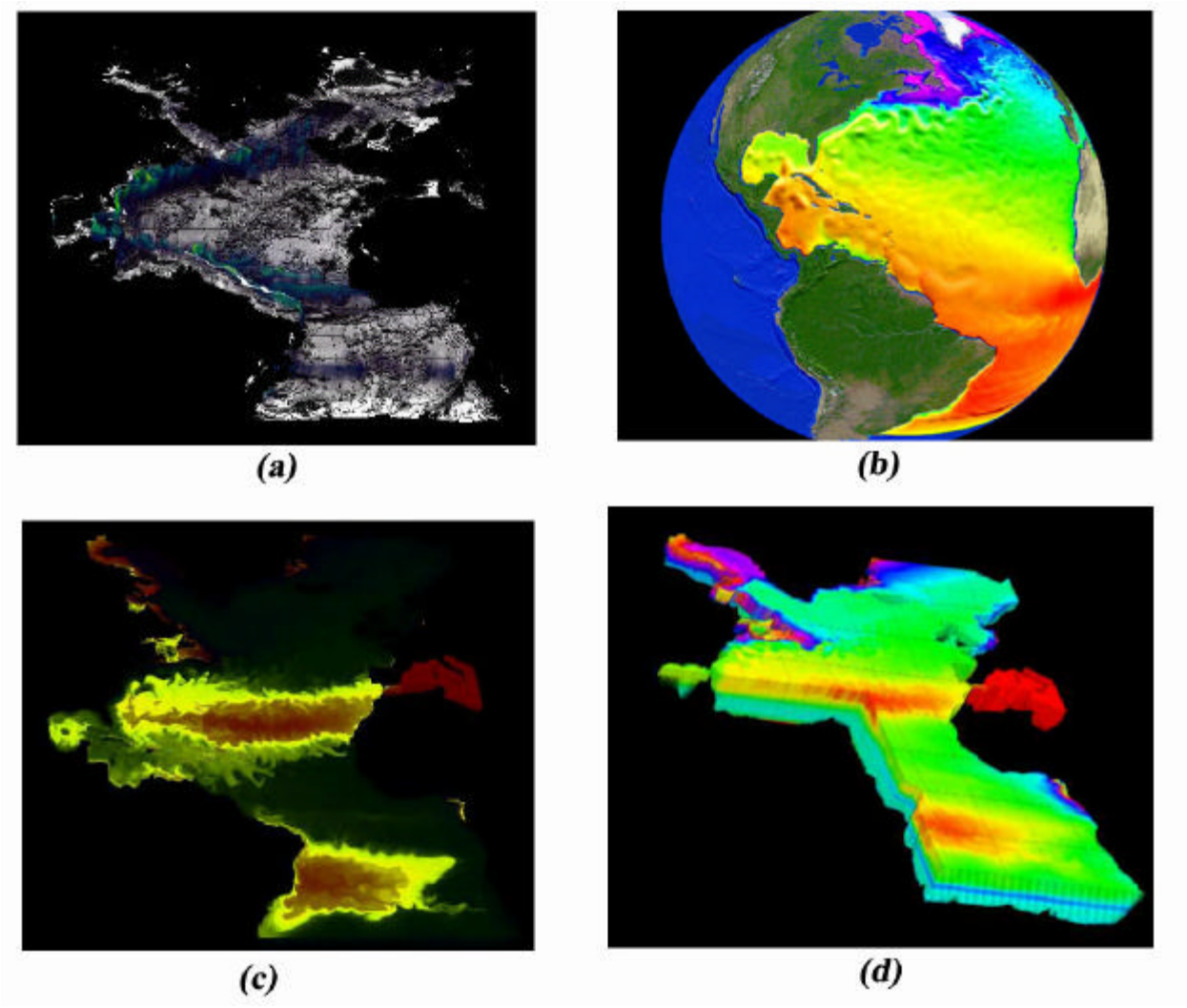

Fig. 5 North Atantic Ocean (a) the velocity volume with the ocean bottom map, (b) ocean surface temperature shaded relieved by the surface height, (c) salinity volume with red for high concentration, (d) using slicing to cut through the salinity volume.

spectrum colormap with purple for low temperature (0? Celsius) and red for high temperature (32? Celsius). The image is then shaded relieved by the surface height data. The height variation is very visible around the north coast. The combined dataset is then wrapped around a global earth image dataset. The ocean surface has the opacity set to 1. In this image, we can see the separation of Gulf Stream off the coast of Cape Hatteras which is not detectable in a lower resolution ocean model.

Fig. 5a shows the magnitude of the ocean velocity vector volume. The transfer function was set such that only the high value has non-zero opacity thus highlighting the high velocity area. It is clear that most of the Atlantic ocean is calm except for the coastal area. The white background is the ocean bottom topography that provides a reference for this mostly translucent image. In a 5-year animation of the ocean velocity, it is clear to see the progression of the mesoscale eddies in the Caribbean Sea. Fig. 5c and Fig 5d are two different representation of the salinity volume. Salinity is another driving force of ocean energy transportation. The high saline water in the Mediterranean Sea (shown in red) is trying to escape to the Atlantic Ocean via the narrow Gibraltar strait. Fig. 5c uses a translucent volume to highlight the high salinity area and Fig. 5d cut out a quarter of the volume using the slicing capability of ParVox in order to see the salinity changes in the lower depth of the ocean.

We produced animations for all the variables shown in Fig. 5. The animations help scientists to understand the major features and the evolution of the Caribbean Sea eddies in 3D space. It also give scientist an effective visual tool to prevail their scientific results to the general public.

V. CONCLUSION

We have presented two parallel rendering systems, RIVA for terrain visualization and ParVox for volume rendering in this paper. Although designed for different applications, the two systems share the same design philosophy:

?? ***Scalability*** – Both ParVox and RIVA are scalable to input dataset size, output image size, and the machine size. With the built-in out-of-rendering capability, there is no limit to the size of the data they can render. Hardware renderers are limited to its physical capacities, such as graphic memory and texture memory. Once the problem size is beyond its hardware limit, it either has serious performance degradation or simply cannot handle it.

?? ***Flexibility*** – ParVox and RIVA were built to address application's need. Each dataset has its unique characteristics and may require a different way to visualize it. Our systems are equipped with a rich set of rendering modules and a user-friendly GUI for a user to pick and choose the best rendering parameters for his dataset.

?? ***Distributed Visualization*** – Distributed visualization is the core design concept for ParVox and RIVA. Our systems are optimized not only in computation, but also in end-to-end frame delivery over low speed network.

We are currently applying RIVA and ParVox to visualize the Earthquake simulation datasets. Similar to the ocean model, the earthquake datasets contain both surface data such as surface deformation and 3D volume such as stress field. Hopefully by the time of the conference, some of the preliminary results can be presented.

ACKNOWLEDGMENT

The work presented in this paper was sponsored by NASA Earth Science Technology Office (ESTO), Computational Technologies Project, formerly known as the Earth and Space Science Project (ESS). The ocean data sets presented was generated by Yi Chao of JPL. I would like to thank James Tsiao, Scott Whitman, William Duquette, and Dave Curkendall for their contribution in the design and implementation of ParVox and RIVA systems. I also like to thank JPL's Supercomputing Project for providing the computer resource and technical support. Finally, I would like to thank Robert Ferraro, my project manager, for his trust and his technical guidance in all these years.

REFERENCES

[1] P. Li, W.H. Duquette, and D.W. Curkendall, "RIVA: A Versatile Parallel Rendering System for Interactive Scientific Visualization," *IEEE Transactions on Visualization and Computer Graphics*, Vol2, No.3, pp 186-201, 1996

[2] P. Li, S. Whitman, R. Mendoza, J. Tsiao, "ParVox – A Parallel Volume Rendering System for Distributed Visualization," *1977 Proceedings of IEEE Symposium on Parallel Rendering*, pp.7-14, 1997

[3] K. Ma and T.W. Crockett, "A Scalable Parallel Cell-Projection Volume Rendering Algorithm for Three-Dimensional Unstructured Data," *1977 Proceedings of IEEE Symposium on Parallel Rendering*, pp.95-104, 1997

[4] E. Majani, "ERIC: An Algorithm for Efficient Reversible Image Compression," *JPL New Technology Report 20141/9777*, January 1977.

[5] "AHPCRC Developed Software:Icol", *http://www.arc.umn.edu/software/bob/icol.html.*

[6] Y. Chao, P. Li, P. Wang, D. Katz, N. Cheng, and S. Whitman, "Ocean Modeling and Visualization on Massively Parallel Computers," *Industrial Strength Parallel Computing: Programming Massively Parallel Processing Systems*, Morgan Kaufmanns, 1998.

[7] R.D. Smith, J.K. Dukowicz, and R.C. Malone, "Parallel Ocean General Circulation Modeling," *Physica D*, 60, 38-61, 1992.